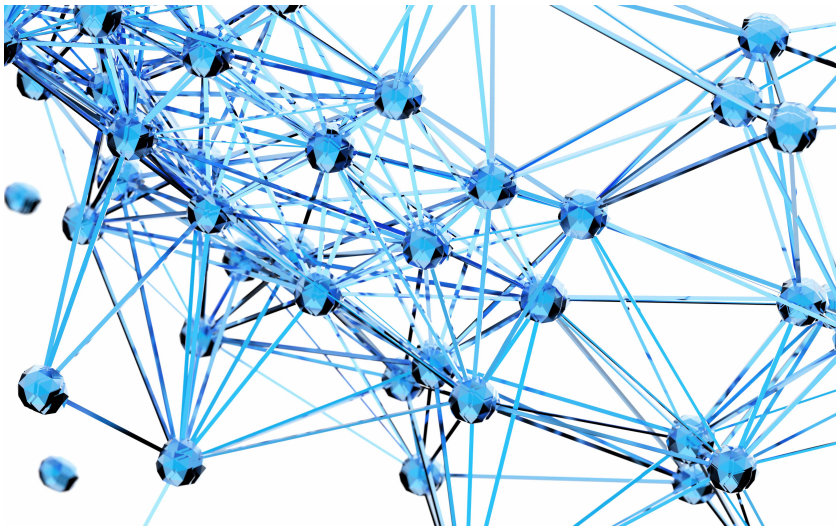
COMP 251 - Network Flows

Algorithms and Data Structures

Written By

David Knapik

McGill University Department of Mathematics and Statistics
david.knapik@mcgill.ca



March 25, 2018

Contents

1 Introduction

Lets say that a good is produced at some source vertex s. We must send as much of the good as possible to a sink vertex t. We have that each arc $a = (i, j)$ has a capacity $u_a = u_{ij}$; the maximum amount of the good that can be sent via that arc (say over a given time period).

Definition 1.0.1. A **network flow** $\vec{f}$ from a source s to a sink t satisfies two properties:

1. Capacity Constraints - the flow on an arc a is non-negative and at most its capacity. I.e. $0 \leq f_a \leq u_a$.
2. Flow Conservation - the flow into a vertex $v \neq \{s, t\}$ equals the flow out of the vertex. I.e. $\sum_{a \in \delta^-(v)} f_a = \sum_{a \in \delta^+(v)} f_a$

Definition 1.0.2. The **value** of a flow $\vec{f}$ is the quantity of flow that reaches the sink t. I.e.

$$value = |f| = \sum_{a \in \delta^-(t)} f_a = \sum_{a \in \delta^+(s)} f_a$$

So the fundamental problem we consider is to find an $s - t$ flow of maximum value. I.e., we want the $\max_{a \in \delta^-(t)} f_a$ such that $\sum_{a \in \delta^-(v)} f_a = \sum_{a \in \delta^+(v)} f_a$ for all $v \in V \setminus \{s, t\}$ and $0 \leq f_a \leq u_a$ for all $a \in A$.

1.1 The Flow Decomposition Theorem

1.1.1 Paths

What do $s - t$ paths have to do with $s - t$ flows? Well, an $s - t$ path is just a flow of value one. So, the union of a set of $s - t$ paths still satisfies flow conservation. And, if in addition this union of paths satisfies the capacity constraint $f_a \leq u_a$ on each arc, then they form an $s - t$ flow.

1.1.2 Cycles

What do directed cycles have to do with $s-t$ flows? Well, a directed cycle also satisfies flow conservation. Thus, the union of a set of cycles satisfies flow conservation. So, if in addition this unions of cycles satisfies the capacity constraint on each arc, then they form an $s - t$ flow.

1.1.3 Putting it Together

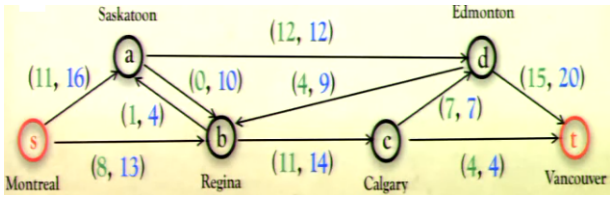
So, from the above, we conclude that the union of a set of paths and a set of cycles satisfies flow conservation. So, if this union also satisfies the capacity constraint on each arc, then we have an $s - t$ flow. We note that the converse is true, namely that every $s - t$ flow is made up of $s - t$ paths and cycles.

Theorem 1.1.1 (Flow Decomposition Theorem)**.** *Any $s-t$ flow can be decomposed into a collection of $s-t$ paths and directed cycles.*

Remark. The decomposition in the Flow Decomposition Theorem need not be unique.

1.2 An Example - Trucking Syrup

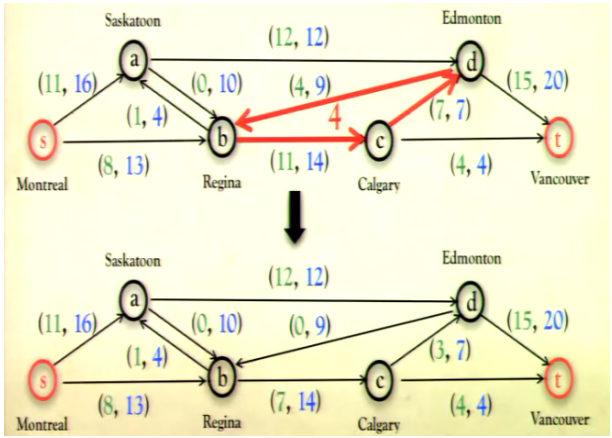
Suppose we send maple syrup on a network from Montreal to Vancouver. For each link a in the network, let u_a be the trucking capacity in tons on the link and f_a be the amount of syrup we are transporting on that link.



Note that we do indeed have an $s-t$ flow with flow value of 19 tons.

1.2.1 An Observation

If a flow puts positive weight on a directed cycle, we can remove this weight from the cycle and still have a flow of the same value.



So, for any $s-t$ flow, there is an $s-t$ flow of the same value that contains NO directed cycles. So combining this observation with theorem 1.1.1, we have that there is a max flow whose flow decomposition contains only directed paths.

2

Ford-Fulkerson Algorithm

2.1 Greedy Approach

Our first attempt at finding a max flow is the greedy approach.

Algorithm 2.1.1 (GreedyMaxFlow)**.**

1. Find a directed path P from s to t
2. Send as much flow on the path P as possible
3. Repeat until there is no surplus capacity on any $s-t$ path

This Greedy Algorithm does not work:

2.1.1 Example

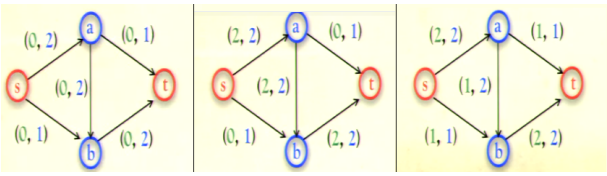
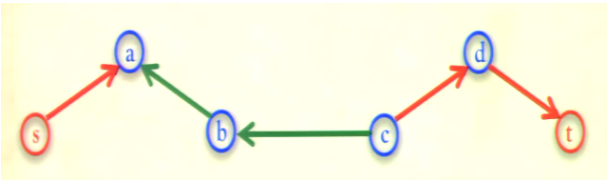


Figure 2.1: At the left is the start. The middle is the result of the Greedy Max Flow Algorithm. The right is a max flow.

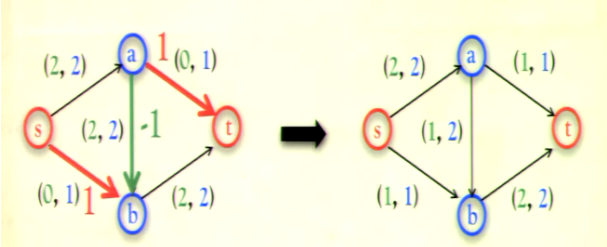
2.2 Augmentation Paths

Definition 2.2.1. An $s-t$ **augmenting path** is a path P from s to t where some of the arcs can be in the reverse direction.

Example 2.2.1.



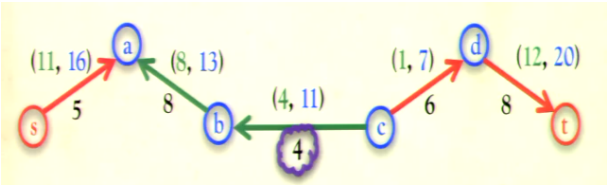
Suppose we increase the flow on a forward arc but decrease the flow on a backward arc on the path P. Then, we actually have that flow conservation is still maintained.



So, using a backwards arc corresponds to reducing the amount of flow used on that arc in the forward direction. So we are correcting a mistake we made earlier.

Now, we know that using an augmenting path has two nice properties. Namely, it maintains flow conservation and increases flow value. But can we ensure that the capacity constraints remain

satisfied? In particular, can we guarantee that $f_a \geq 0$ isn't violated? Well, given the current flow $\vec{f}$, we can increase the flow on a forward arc (i, j) by $u_{ij} - f_{ij}$. And, we can decrease the flow on a backward arc (i, j) by f_{ij}.



In particular, we can increase the flow on the augmenting path P by the bottleneck capacity of P w.r.t flow $\vec{f}$:

$$b(P, \vec{f}) = \min \left[\min_{(i,j)\ forward\ arc} u_{ij} - f_{ij}, \min_{(i,j)\ backward\ arc} f_{ij} \right]$$

So, we have derived the Ford-Fulkerson Algorithm:

Algorithm 2.2.1 (Ford-Fulkerson)**.** Set $\vec{f} = 0$. Repeat the following:

1. Find an augmenting path P w.r.t $\vec{f}$
2. Augment flow on path P by $b(P, \vec{f})$

2.3 The Residual Graph

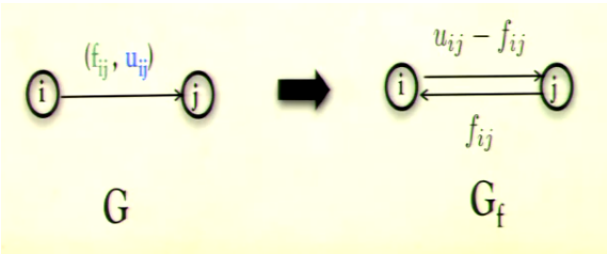
In this section, we examine the key to both implementing and understanding the Ford-Fulkerson Algorithm - residual graphs. With this tool, we will restate the Ford-Fulkerson algorithm.

Recall that in the Ford-Fulkerson algorithm , we had to use arcs in both the forward and backward directions. We are allowed to use arcs in the forward direction if the arc $a = (i, j)$ has spare capacity; $f_{ij} < u_{ij}$. We are allowed to use arcs in the backward direction if the arc $a = (i, j)$ has already been used forwards and thus flow can be sent back (so need $f_{ij} > 0$).

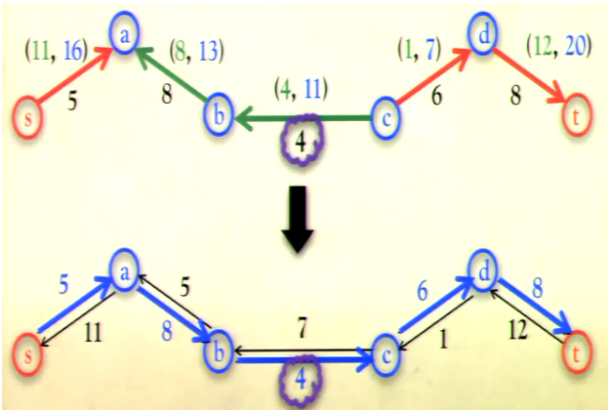
Definition 2.3.1. Given $G = (V, A)$ and a flow $\vec{f}$, we define the residual graph G_f as follows: For each arc $a = (i, j) \in A$

- G_f contains an arc (i, j) of capacity $u_a - f_a$
- G_f contains an arc (j, i) of capacity f_a

Example 2.3.1.



Observe that the arcs in the residual graph only have capacities. The bottleneck capacity of a path is now just the minimum capacity of an arc in the corresponding directed path in the residual graph.



So, we restate the Ford-Fulkerson Algorithm:

Algorithm 2.3.1 (Ford-Fulkerson v2)**.** Set $\vec{f} = 0$. Repeat the following:

1. Find a directed $s - t$ path P in the residual graph G_f
2. Augment flow on path P by its bottleneck capacity $b(P, \vec{f})$

Example 2.3.2. See lecture recording. Understand how this algorithm works to find a max flow!!!

Remark. We note that the Ford-Fulkerson algorithm terminates when there are no $s - t$ paths left in the residual graph.

2.4 Maxflow-Mincut Theorem

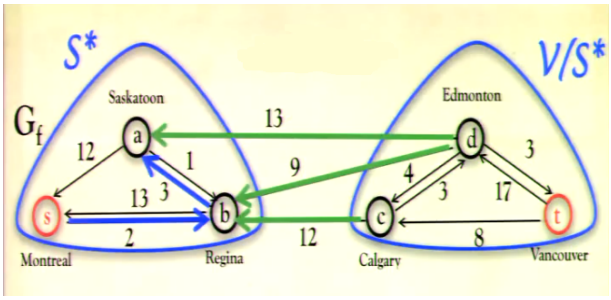
Given our newly updated Ford-Fulkerson algorithm, we would like to prove it works, finds a max flow, and we would like to examine its efficiency.

Definition 2.4.1. We say that $(\mathcal{S}, V \setminus \mathcal{S})$ is an $s - t$ cut if $s \in \mathcal{S}$ and $t \notin \mathcal{S}$

Let $\vec{f}^*$ be the flow output when the Ford-Fulkerson algorithm terminates. Let $\mathcal{S}^*$ denote the set of vertices reachable from s in the residual graph G_{f^*} i.e.

$$\mathcal{S}^* = \{v : \exists \text{ directed path from } s \text{ to } v \text{ in } G_{f^*}\}$$

Note that $(\mathcal{S}^*, V \setminus \mathcal{S}^*)$ is an $s - t$ cut (since trivially $s \in \mathcal{S}^*$ and by the termination condition, $t \notin \mathcal{S}^*$). Also, there are no arcs leaving $\mathcal{S}^*$ in G_{f^*} (if not then we could add an extra vertex to $\mathcal{S}^*$).



Lemma 2.4.1 (The Cut Lemma)**.** Given a flow $\vec{f}$ and an $s - t$ cut $(\mathcal{S}, V \setminus \mathcal{S})$, the value of the flow is

$$|f| = \sum_{a \in \delta^+(s)} f_a - \sum_{a \in \delta^-(s)} f_a$$

Proof. The value of the flow is the amount of flow leaving the source s, so

$$\begin{aligned}
|f| &= \sum_{a\in\delta^+(s)} f_a \\
&= \sum_{a\in\delta^+(s)} - \sum_{a\in\delta^-(s)} f_a \quad \text{the second sum we introduce is 0 because } \delta^-(s)=\emptyset \\
&= \left(\sum_{a\in\delta^+(s)} - \sum_{a\in\delta^-(s)} f_a\right) + \sum_{v\in\mathcal{S}\setminus\{s\}} \left(\sum_{a\in\delta^+(v)} f_a - \sum_{a\in\delta^-(v)} f_a\right) \\
&= \sum_{v\in\mathcal{S}} \left(\sum_{a\in\delta^+(v)} f_a - \sum_{a\in\delta^-(v)} f_a\right)
\end{aligned}$$

(where we have used that $(\sum_{a\in\delta^+(v)} f_a - \sum_{a\in\delta^-(v)} f_a)$=0 via flow conservation). Now, take any arc $a=(i,j)\in A$ in the graph. There are four cases we need to deal with.

1. If $(i,j)\in\delta^+(\mathcal{S}^*)$ then f_a appears once in the sum with coefficient 1
2. If $(i,j)\in\delta^-(\mathcal{S}^*)$ then f_a appears once in the sum with coefficient -1
3. If $\{i,j\}\subset V\setminus\mathcal{S}^*$ then f_a does not appear in the sum.
4. If $\{i,j\}\subset\mathcal{S}^*$ then f_a appears twice in the sum with coefficients -1 and 1. So , we have cancellation.

Therefore, $|f| = \sum_{a\in\delta^+(s)} f_a - \sum_{a\in\delta^-(s)} f_a$ □

Definition 2.4.2. We define the capacity of an $s-t$ cut $(\mathcal{S}, V\setminus\mathcal{S})$ as

$$Cap(\mathcal{S}) = \sum_{a\in\delta^+(\mathcal{S})} u_a$$

Lemma 2.4.2. Given any flow $\vec{f}$ and any $s-t$ cut $(\mathcal{S}, V\setminus\mathcal{S})$ we have that

$$|f| \le Cap(\mathcal{S})$$

Proof. Using the Cut Lemma, we have

$$\begin{aligned}
|f| &= \sum_{a\in\delta^+(s)} f_a - \sum_{a\in\delta^-(s)} f_a \\
&\le \sum_{a\in\delta^+(\mathcal{S})} u_a + 0
\end{aligned}$$

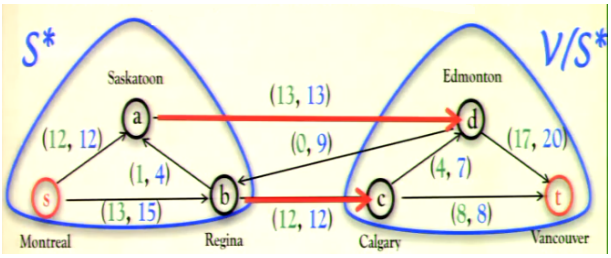
□

So, in particular, the value of a max flow is at most the capacity of the minimum cut (cut with the minimum capacity). In fact, they are equal:

Theorem 2.4.1 (Maxflow-Mincut)**.** *The max value of an $s-t$ flow is equal to the minimum capacity of an $s-t$ cut.*

Proof. By the above lemma, we have that $|f^*| \le Cap(\mathcal{S}^*)$. Thus, we need only to show that $|f^*| \ge Cap(\mathcal{S}^*)$.

Now, recall that there are no arcs leaving $\mathcal{S}^*$ in G_{f^*}. What does this imply about the flow $\vec{f}^*$? Well, take any arc $(i,j)\in\delta^+(S^*)$ (this arc is not in the residual graph). Then ${f_{ij}}^* = u_{ij}$.



On the other hand, take any arc $(i, j) \in \delta^-(\mathcal{S}^*)$. This arc is not in the residual graph. Then, ${f_{ij}}^* = 0$.

So, by the Cut Lemma, we have:

$$\begin{aligned} |f^*| &= \sum_{a \in \delta^+(\mathcal{S}^*)} f_a - \sum_{a \in \delta^-(\mathcal{S}^*)} f_a \\ &= \sum_{a \in \delta^+(\mathcal{S}^*)} u_a - \sum_{a \in \delta^-(\mathcal{S}^*)} f_a \\ &= \sum_{a \in \delta^+(\mathcal{S}^*)} u_a - 0 \\ &= Cap(\mathcal{S}^*) \end{aligned}$$

□

Immediately from the Maxflow-Mincut Theorem, we have that the Ford-Fulkerson algorithm works and finds a max flow. Is it efficient though? Well, we can find G_f in $O(m)$ and we can find an $s - t$ path P in G_f in $O(m)$. Finally, we can augment the flow on P in $O(n)$. So, the run-time is $O(m \cdot N)$ where N is the number of iterations. To find N, recall that the algorithm terminates when $b(P, \vec{f}) = 0$ for every $s - t$ path P. Thus, since the capacities are integral, we have that $b(P, \vec{f}) \geq 1$ while running. So, in the worst case, we increase the flow value by only 1 each iteration. By the Maxflow-Mincut theorem, the max flow equals the min cut value C. If $U = \max_a u_a$ then $C \leq U$, so the runtime is

$$O(mn \cdot U)$$

which is only pseudo-polynomial. Recall that this is not truly polynomial time in input size. For example, if the integers have b bits then U could be 2^b.

Acknowledgements

These notes have been transcribed from the lectures given by Prof. Adrian Vetta. Any figures are screenshots of the recorded lectures. This document is for personal use only and beware of typos!